

## AN ALGORITHM FOR ASSEMBLING OVERLAPPING GRID SYSTEMS\*

N. ANDERS PETERSSON<sup>†</sup>

**Abstract.** A general purpose algorithm for assembling overlapping grid systems for solving partial differential equations (PDEs) is described. The method combines and improves on the techniques implemented in the codes Beggar, CMPGRD, DCF3D, and PEGSUS. The present algorithm, which has been implemented for the two-dimensional case in the code Xcog, starts by calculating a global definition of the boundary of the computational domain based on the location of the physical boundaries in all component grids. The global boundary description is used to identify all grid points outside of the computational domain. The remaining points are classified according to the priority of the component grids in the overlapping grid. If the overlap between the components is sufficiently large, all remaining points can either interpolate from an overlapping grid or be used to discretize the PDE. The algorithm can therefore construct an overlapping grid for implicit (coupled) interpolation in one pass and needs only to iterate to ensure explicit (decoupled) interpolation. If the algorithm fails due to insufficient overlap between the components, the main parts of the grid will still be valid. This enables the code to report inconsistent grid points to the user, to facilitate an improvement of the input.

Discrepancies between the boundary representations where the grids overlap are handled by a mismatch tolerance, which is estimated automatically. The interpolation data are corrected for the boundary mismatch, and it is shown how the interpolation error in a thin boundary layer is substantially reduced by this correction.

**Key words.** overlapping grid, overset grid, Chimera grid

**AMS subject classification.** 65M50

**PII.** S1064827597292917

**1. Introduction.** The overlapping grid method, also known as the Chimera overset grid technique, provides a flexible and efficient spatial discretization procedure for numerically solving a PDE on a general two- or three-dimensional domain. In this paper, we discuss the construction of overlapping grids and present a new algorithm for the grid assembly.

An overlapping grid consists of a number of component grids, where boundary-fitted curvilinear components resolve the different details close to the boundary of the computational domain and where the background grids, which are often Cartesian, cover the remaining parts of the domain; see Figure 1.1. Each component grid is structured and can have three types of grid points: discretization, interpolation, and hole points. The discretization points are used to discretize the PDE or the boundary conditions; the interpolation points interpolate their solution value from the overlapping component grid; and the hole points are disregarded during the discretization of the PDE. The hole points are either outside of the computational domain or are eliminated to reduce the total number of grid points in the overlapping grid.

An overlapping grid can be constructed from a set of component grids if they overlap each other sufficiently and if the sides<sup>1</sup> of the components that describe the physical boundary are identified. The basic overlap algorithm, developed by Benek,

---

\*Received by the editors March 31, 1997; accepted for publication (in revised form) October 8, 1997; published electronically June 3, 1999.

<http://www.siam.org/journals/sisc/20-6/29291.html>

<sup>†</sup>Hydromechanics Division, Naval Architecture and Ocean Engineering, Chalmers University of Technology, 412 96 Gothenburg, Sweden (andersp@na.chalmers.se).

<sup>1</sup>A side of a component grid is a grid surface in three dimensions and a grid line in two dimensions.

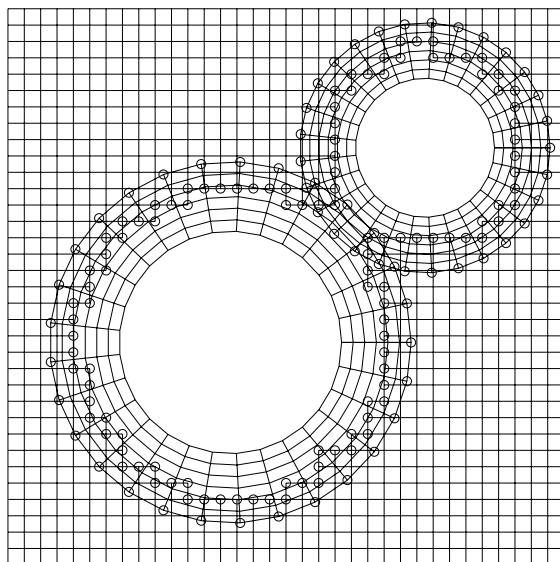


FIG. 1.1. A simple overlapping grid. The circles indicate interpolation points where the solution value is interpolated from the overlapping component grid.

Steger, and Dougherty [2] and Kreiss [6], works well when each part of the boundary is completely described by one side of one component grid, as in Figure 1.1. The method consists of two major steps.

The first step is to detect all hole points outside of the computational domain, and the second step is to find the grid points to interpolate from (the donor points) for all interpolation points on the fringe of the hole. Since each part of the boundary is completely described by one closed grid line, it is possible to use this grid line to determine whether a point in the background grid is inside or outside the computational domain.

It becomes harder to detect the hole points when more than one component grid describes each part of the boundary of the computational domain, as in Figure 1.2. In this case, there is no closed grid line that can be used to cut out holes from the background grids. Furthermore, there can be a mismatch at the boundary where the components overlap each other. This makes the definition of the boundary slightly imprecise, which complicates the hole-cutting process since there is no unique definition of the extent of the computational domain close to the boundary in the overlap region.

The boundary mismatch also hampers the selection of interpolation points and donor points. When the surface is convex, it becomes difficult to select the right donor points for interpolation points on physical boundaries, because they appear to be slightly inside of the boundary in the donor grid. Also, when the surface is concave, boundary interpolation points appear to be slightly outside of the boundary in the donor grid, so the boundary mismatch can lead to the improper classification of those interpolation points as hole points.

The mismatch is especially likely to happen on curved boundaries that are not very well resolved by the grid, when the boundary is approximated by a linear interpolant between the discrete grid points; see Figure 1.3. However, the mismatch can also occur

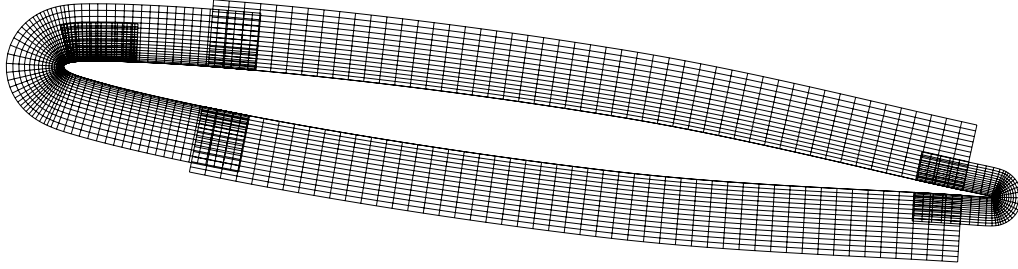


FIG. 1.2. The boundary-fitted components in an overlapping grid around a NACA-66-006 airfoil. Several components are used next to the airfoil to optimize the grid sizes by making the grid fine only where the gradient of the solution is large.

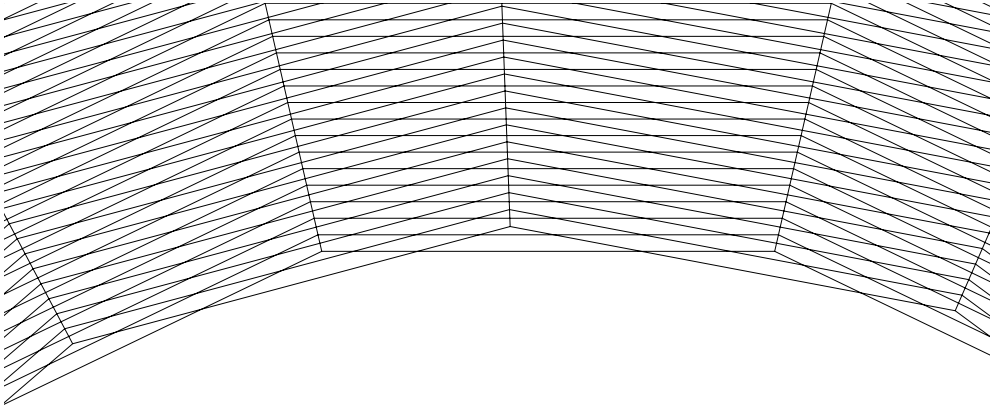


FIG. 1.3. A mismatch between the representations of a physical boundary can occur on curved boundaries. In this case, the circular boundary is approximated by a linear interpolant between the discrete grid points.

when a smoother approximation of the boundary is used and if the representations of the boundary are different in the grids that overlap each other.

The mismatch problem becomes more pronounced when the grid is very fine in the direction normal to the boundary and when the grid cells have a large aspect ratio. If the mismatch is of the same magnitude or larger than the grid size in the direction normal to the physical boundary, the grid points on the physical boundary in one component can be situated several cells inside or outside of the physical boundary in the overlapping component. Hence, there can be a large error in interpolated solution

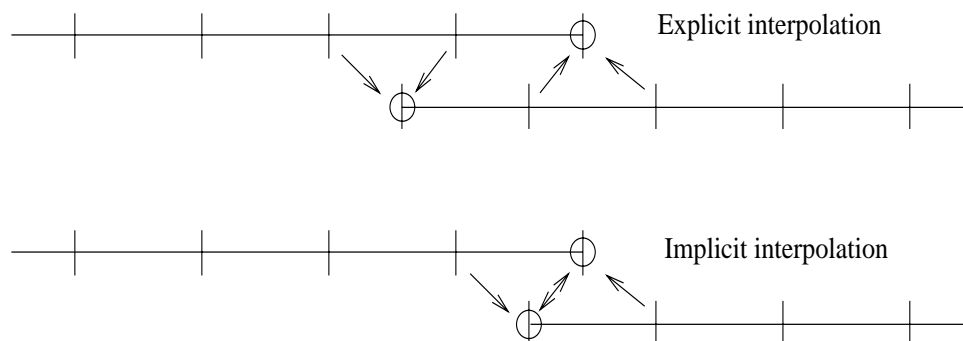


FIG. 1.4. *Explicit and implicit interpolations for a one-dimensional overlapping grid.*

values if the solution has a boundary layer.

There are two additional practical complications related to making an overlapping grid. First, it can be hard to a priori judge if the component grids overlap each other sufficiently. Second, the user can make a mistake when labeling the physical sides of the component grids, which can lead to an inconsistent definition of the boundary of the computational domain. Creating an overlapping grid is therefore sometimes an iterative process, where the component grids are changed by the user until a valid overlapping grid can be formed. Hence, it is important that the overlap algorithm be fast so that the turnaround time for the grid assembly is small. It is also desirable that the overlap algorithm produce a helpful error message when it fails, to facilitate an improvement of the input. We remark in passing that it is also important that the overlap algorithm be fast during a moving grid simulation, where the overlap information has to be updated at every time-step. However, moving grids will not be discussed further in the present paper.

There are two different ways to interpolate in an overlapping grid; see Figure 1.4.

When the interpolation type is implicit, the solution values at the interpolation points are coupled, because they interpolate from both discretization and interpolation points in the donor grid. This makes the required overlap smaller compared to when explicit interpolation is used, since in that case only discretization points are allowed to be donor points. Explicit interpolation is sometimes preferred when a time-dependent problem is solved on the overlapping grid, because it simplifies the solution procedure. Explicit interpolation can also be used for elliptic problems, where it can be useful in connection with domain decomposition techniques. Implicit interpolation is well suited for solving elliptic equations by a direct method such as Gaussian elimination. In this case the interpolation relations are additional linear equations that are solved simultaneously with the discretized elliptic equation on each component grid. Implicit interpolation can also be used to save grid points both for steady-state and time-accurate simulations of time-dependent PDEs. However, in the latter case it becomes necessary to solve a linear system of equations to update the solution values at all interpolation points after each time-step.

The basic overlap algorithm must be refined to construct an optimal overlapping grid, where the overlap is as small as possible. When the interpolation is implicit, the component grids must overlap each other by at least half a grid cell (for linear or quadratic interpolation). Furthermore, the required amount of overlap is independent of the number of component grids that overlap each other. The situation is different

for explicit interpolation. For instance, if the discretization and interpolation stencils are three points wide in each grid direction, the amount of overlap must exceed  $1\frac{1}{2}$  grid cells where two components overlap each other. Also, the overlap must be up to three grid cells wide close to where more than two grids overlap each other.

**1.1. Previous methods.** A number of algorithms exists for constructing overlapping grids. The technique developed by Benek, Buning, and Steger [1] and Benek, Steger, and Dougherty [2] has been implemented and developed further in the code PEGSUS [17]. The algorithm begins by cutting out holes from grids where the grid points are outside of the computational domain and proceeds by finding donor points for the interpolation points, which are located on the fringes of the holes and on interpolating sides of the component grids.

In order for the algorithm to cut the holes, the user must specify the hole-cutting surface as well as the hole grid, i.e., the grid in which the hole should be made. The hole-cutting surface can be either one or more grid surfaces, or one or more rectangular boxes. Each grid point in the hole grid is checked to see if it is inside or outside of the hole-cutting surface. When the hole-cutting surface is a collection of grid surfaces, the algorithm traverses through every grid point on every grid surface and compares the angle between the outwardly directed normal to the grid surface, and the vector from the grid point on the surface to the grid point in the hole grid. This approach requires the collection of grid surfaces to be convex, so holes with a concave boundary must be decomposed into convex parts by the user before the algorithm is applied.

In PEGSUS, the location of physical surfaces is defined by linear interpolation between the discrete grid points, which results in a mismatch problem. To remedy the situation, Parks et al. [12] suggest that the interpolation points on the physical boundary be moved to coincide with the description of the discrete boundary of the donor grid. Thereafter, either the grid points inside the surface are moved the same distance or the corresponding component grids are regenerated using the new boundary description. This straightforward approach requires the overlap along the boundary to be sufficiently large to make the boundary interpolation decouple. Furthermore, if the component grids are regenerated, the method adds an extra step to the overlap algorithm.

A method related to PEGSUS was proposed by Meakin [9], where three improvements to speed up the algorithm were introduced. First, hole-cutting surfaces are defined by combinations of simple analytical shapes such as cylinders and spheres. Although this imposes restrictions on the shape of the object to be gridded, it makes it straightforward and inexpensive to determine whether a grid point in the major grid is inside or outside of the hole-cutting surface. The second way to speed up the grid generation process is by using inverse maps. These maps are Cartesian helper grids that cover the curvilinear grids. The parameter coordinate of the curvilinear grid is precomputed at each vertex of the Cartesian grid and is used for finding donor cells for the interpolation points on the fringes of the holes. The third approach to accelerate the overlap algorithm is to let the inverse maps move together with the component grids during a rigid body motion. This enables the overlap information to be quickly updated, which is advantageous during a moving grid computation. This overlap algorithm is implemented in the code DCF3D.

A combination of the overlapping grid method and the patched multiblock approach, where the component grids have common internal boundaries, is used in the technique developed by Maple and Belk [8]. In this method, there can be one or more patched component grids within each superblock. The superblocks overlap each other

and communicate through interpolation. Hence, the combined grid can be entirely patched with only one superblock, or completely overlapping with one component per superblock, or somewhere in between. A data structure based on a combination of octrees and binary space partition (BSP)-trees is used for determining whether a point is inside or outside of a superblock as well as for providing an initial approximation for locating donor points. The algorithm uses all grid surfaces with solid boundary condition to cut holes. The hole points outside of the computational domain are identified in a two-step mark and fill process, which requires the hole-cutting grid surfaces to be closed, but allows very thin holes. After the holes have been made, the algorithm compiles a list of potential donor points for each interpolation point on the fringes of the holes. The algorithm locates the donor that is considered to give the best interpolation, and if all candidates are unfit as donor points, the interpolation point is reclassified into a hole point, and the fringe of the hole is moved. The selection of interpolation and donor points is done iteratively, since there might be interpolation points in other grids that used an interpolation point as a donor point before it was reclassified into a hole point. This algorithm is implemented in a code called Beggar.

Similar to PEGSUS, the Beggar code uses a linear approximation of physical boundaries between the discrete grid points in each component grid. To circumvent the resulting mismatch problem, Noack and Belk [11] suggest that a global reference geometry be used to define the location of the physical surface between the discrete grid points. The error between the linear surface representation and the reference geometry is calculated as a function of the surface coordinates. This error is accounted for when donor points are selected and interpolation weights are computed. In their implementation, a piecewise linear representation which includes all boundary grid points in all components that are aligned with the same surface is used as the reference geometry. In principle, a more accurate CAD-based description could be used instead.

To summarize, the basic approach of PEGSUS, DCF3D, and Beggar is to first detect the hole points and then find donor points for the interpolation points on the fringe of the hole and on interpolating sides of the component grids. Holes are made in grids where they are outside of the hole-cutting surfaces, so optimal overlap or explicit interpolation can be ensured only by placing those surfaces carefully. Furthermore, both PEGSUS and DCF3D require rather detailed user input concerning where holes should be cut from the component grids.

A more automatic method was developed by Chesshire and Henshaw [4] and originally implemented in the Fortran code CMPGRD [3]. The current version of the code is rewritten in C++ and is part of the Overture framework [5]. In this method, the component grids are ordered by priority. The algorithm determines iteratively where the holes should be made, based on where the physical boundary is located. Furthermore, it constructs an optimal grid for implicit or explicit interpolation. The complications resulting from boundary mismatch are mitigated by using a mapping for each component grid to define the behavior of the grid in between the discrete grid points. Discrepancies between the boundary representations where the mappings overlap are handled by a user specified mismatch tolerance.

The algorithm in CMPGRD begins by traversing through all grid points situated on physical boundaries to mark the nearest neighboring grid points in all other components as hole points. The classification of the remaining points is done iteratively. In each iteration, the algorithm passes sequentially through all grid points in all component grids. For each grid point, it examines whether it can interpolate from a grid with higher priority, be a discretization point, or interpolate from a lower grid. If none of the above are possible, the grid point is labeled as a hole point. The

algorithm iterates on the classification of all points until no more changes are made. Since all points that are next to a physical boundary point in another grid are initially marked as hole points, and since there cannot be a discretization point next to a hole point, the holes will grow during the iteration to exclude all points that are outside of the computational domain. Unnecessary interpolation points are then trimmed away to produce an overlapping grid where the overlap between the components is as small as possible.

Although the iterative approach in CMPGRD makes it possible to construct overlapping grids around very complicated objects, where the components may overlap each other arbitrarily, the algorithm encounters difficulties if the overlap between two component grids is too narrow or if the sides of the components that describe the physical boundary are incorrectly identified. In that case, the number of hole points will increase during each iteration of the classification, resulting in an empty overlapping grid where all grid points are labeled as hole points.

**1.2. Outline of the present method.** The present method, which has been implemented for the two-dimensional case in the code Xcog [14], can be seen as a combination of the two above approaches. Similar to PEGSUS, DCF3D, and Beggar, all points that are outside of the computational domain are first marked as hole points. The present method automatically computes a global definition of the boundary of the computational domain based on the location of all physical boundaries in all component grids. Hence, the user input is simpler compared to PEGSUS and DCF3D. Furthermore, the global approach can handle boundaries that are described by several overlapping components and does not require the sides with a solid boundary condition to form closed surfaces. Similar to CMPGRD, the boundary mismatch problem is handled by using a mapping for each component grid together with a mismatch tolerance, which in the present method is estimated automatically. In addition, the present method compensates the interpolation data for the boundary mismatch.

After the identification of the hole points outside of the computational domain, the algorithm in Xcog proceeds by classifying the remaining points according to the CMPGRD method by ordering the components by priority. If the overlap between the components is sufficiently large, all points remaining after the hole-cutting can be used either to discretize the PDE or to interpolate from an overlapping grid. Hence, only one sweep through all points is necessary to construct a grid with implicit interpolation. An iterative reclassification of interpolation points and donor points is applied if the interpolation needs to be explicit. Finally, the grid is trimmed to change unnecessary interpolation points into discretization or hole points to produce a grid with as little overlap as possible. In contrast to CMPGRD, most parts of the overlapping grid will be valid even after the algorithm fails due to insufficient overlap between the components. Any inconsistent grid points can therefore be reported to the user, who can utilize this information to modify the component grids to improve the situation.

The overlapping grid algorithm will be described for a two-dimensional domain and a vertex centered discretization scheme, where the discrete solution values are located at the grid points. While these assumptions make the presentation of the algorithm simpler, they are not restrictions of the overlapping grid method.

The remainder of the paper is organized as follows. We discuss the boundary specification in section 2 and the requirements on the grid points in an overlapping grid in section 3. In section 4, we describe the algorithm for inverting general grid mappings, which is used for finding donor points for the interpolation points. This is

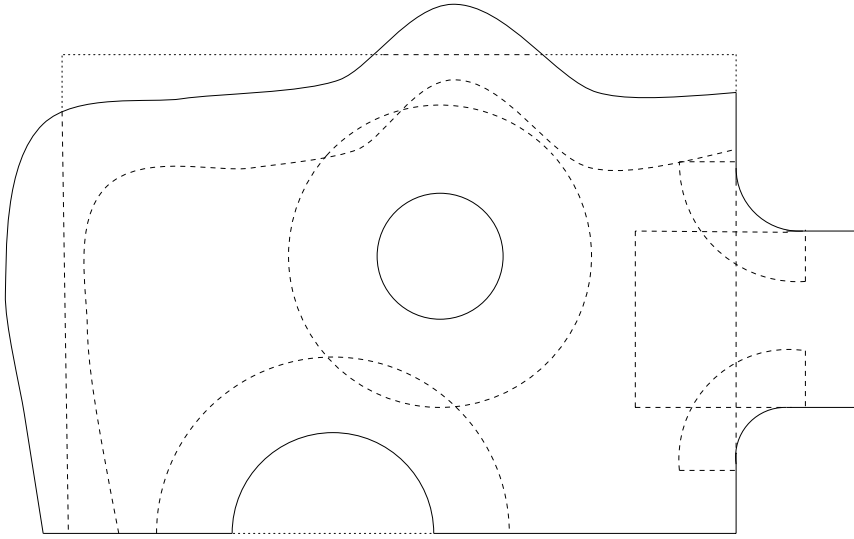


FIG. 2.1. *The computational domain is determined by the physical boundaries and by the position of the component grids relative to those boundaries. Here solid lines are physical boundaries, dashed lines represent interpolation boundaries, and dotted lines indicate external boundaries.*

followed by a presentation of the overlap algorithm in section 5, where each grid point is classified as either a discretization, an interpolation, or a hole point. Finally, in section 6 we consider the potential flow around a circular cylinder augmented by a thin boundary layer. It is demonstrated how the interpolation error caused by boundary mismatch can be substantially reduced by correcting the interpolation data.

**2. Boundary specification.** The overlap algorithm distinguishes between two main types of boundary points in a component grid: physical and nonphysical. The physical boundary points describe the boundary of the computational domain. For example, in a fluid flow model all grid points on no-slip, slip, inflow, outflow, and far-field boundaries are physical boundary points. The nonphysical points can be interpolating or external. An interpolating boundary point lies inside another component grid such that the solution value can be interpolated to that boundary, and an external boundary point is situated outside of the computational domain; see Figure 2.1.

To properly cut holes in the component grids, we use a global description of the boundary of the possibly multiply connected computational domain. The most general way to specify the boundary would be to use a pointwise approach, where the user would be able to input the type for each individual grid point on the boundary. However, the amount of data would then be very large, and there would also be a considerable risk of making input errors that could lead to an inconsistent definition of the boundary. To simplify the input but still allow for flexibility, each side of each component grid is labeled as either physical, mixed physical/interpolating, or nonphysical. There can also be user specified external portions of physical and mixed physical/interpolating boundaries.

The physical/interpolating boundary condition is used for inlet or outlet geometries, where one component is sticking into another component, sometimes with the corners rounded off by fillet grids. Each grid point on the physical/interpolating sides

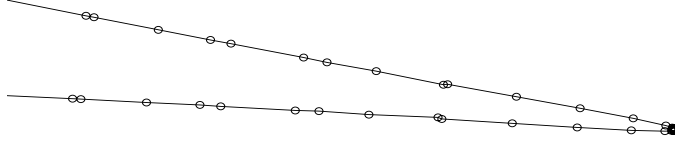


FIG. 2.2. The boundary grid points close to the trailing edge of the NACA-66-006 airfoil shown in Figure 1.2.

that is outside of all other component grids is considered to be a physical point. These sides can therefore not be used to cut holes in other component grids.

In practice, the boundary of the computational domain is identified by assigning a curve label to the sides of the components that are aligned with the boundary. Sides that are completely aligned with the boundary are given a positive curve label, and sides that are partly on the boundary and partly inside of another component are given a negative curve label. The absolute value of the curve label is the same for all sides of all components that are aligned with the same part of the boundary. Hence, if the domain is simply connected, only one curve label value is used. For a doubly connected domain, two values are used, and so on. The external portion of a physical or mixed physical/interpolating side is specified by giving the starting and ending positions of the gap along the boundary.

The global boundary description is computed by sorting the physical points that belong to the same part of the boundary, i.e., have the same absolute value of the curve label. The polygon that joins the sorted points is used to represent the corresponding part of the global boundary. By necessity, each polygon is closed, and it is determined whether the computational domain is on the inside or the outside of the polygon by checking on which side the corresponding component grids are situated.

Some care is required to properly sort the physical boundary points. For instance, the straightforward approach to search for the closest neighbor breaks down for thin bodies when the tangential distance between the points exceeds the thickness of the body; cf. Figure 2.2. To avoid this problem, we treat the distances in the normal and tangential directions differently. Let  $\mathbf{t}$  be the tangent and  $\mathbf{n}$  be the normal of the boundary estimated by using the locations of the current and the previous points. The next point is then selected as the point that has the smallest distance from the current point  $\mathbf{x}_P$  according to the distance function

$$d(\mathbf{x}) = \begin{cases} \sqrt{((\mathbf{x} - \mathbf{x}_P) \cdot \mathbf{t}/a)^2 + ((\mathbf{x} - \mathbf{x}_P) \cdot \mathbf{n})^2}, & (\mathbf{x} - \mathbf{x}_P) \cdot \mathbf{t} \geq 0, \\ \sqrt{(a(\mathbf{x} - \mathbf{x}_P) \cdot \mathbf{t})^2 + ((\mathbf{x} - \mathbf{x}_P) \cdot \mathbf{n})^2}, & (\mathbf{x} - \mathbf{x}_P) \cdot \mathbf{t} < 0. \end{cases}$$

The coefficient  $a$  measures the importance of the tangential distance compared to the normal distance. A value of  $a = 5$  has proven to work well in practice. The case with negative tangential distance is specially treated to avoid alternating direction at the beginning of the polygon. However, it is necessary to account for points with a negative tangential distance, for instance, at sharp corners where the tangent changes direction by more than  $90^\circ$ .

**3. Requirements on the grid points in an overlapping grid.** A grid point is a valid discretization point if it is possible to set up the discretization stencil at

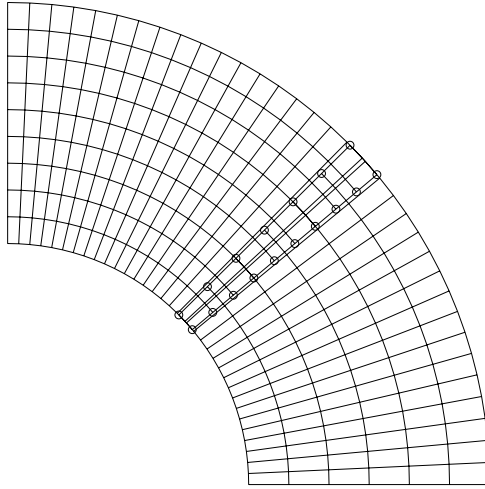


FIG. 3.1. An interpolation point must be separated from nonphysical boundaries in the donor grid, but is allowed to be close to a physical boundary if the interpolation point is close to a physical boundary in its own component grid. Here, the interpolation points are marked with circles.

the point, that is, if there are no hole points in the stencil. The present algorithm allows for different widths for interior, boundary, and corner discretization stencils. Henceforth, we assume that the width of the interior stencil is  $D_W \times D_W$ , where  $D_W$  is odd and satisfies  $D_W \geq 3$ . This implies that each interior discretization point must be separated from all hole points by at least one interpolation point.

For the interpolation points, which interpolate from an  $I_W \times I_W$  stencil in the donor grid, the algorithm must make sure that none of the donor points are hole points. The width of the interpolation stencil is chosen based on the order of accuracy, the type of PDE (elliptic, parabolic, hyperbolic, etc.), and by the behavior of the overlap when the grid size decreases; see [4] for details. In the following, we will assume that  $I_W \geq 2$ .

It is convenient to regard each component grid as a mapping from the unit square in parameter space to the physical domain covered by the component grid. Hence, a grid point is inside of the grid if and only if the corresponding parameter coordinate is inside of the unit square in parameter space. We assume that the mapping for each component grid is one-to-one and that the corresponding grid in parameter space is Cartesian with constant step size. Note that if only the locations of the grid points are known, the mapping can be approximated by interpolation between the grid points to the required order of accuracy. In the following, we will denote a coordinate in parameter space by  $\mathbf{r} = (r, s)$  and a Cartesian coordinate in physical space by  $\mathbf{x} = (x, y)$ . Furthermore, let the mapping for component grid  $k$  be  $\mathbf{x} = \mathbf{X}^{(k)}(\mathbf{r})$ , and let it cover the domain  $\mathbf{x} \in \Omega^{(k)}$ .

Consider one grid point in component  $A$  with Cartesian coordinate  $\mathbf{x}_P$  that is inside of donor grid  $B$ . By inverting the donor grid's mapping function, we can compute the donor parameter value  $\mathbf{r}_P : \mathbf{x}_P = \mathbf{X}^{(B)}(\mathbf{r}_P)$ . To prevent the overlap between components  $A$  and  $B$  from becoming arbitrarily small, the point  $\mathbf{x}_P$  is considered to be a valid interpolation point only if  $\mathbf{r}_P$  is at least  $\max(0.5, 0.5(I_W - 2))$  grid cells inside of all nonphysical boundary points in grid  $B$ . However, no overlap requirements can be enforced in the direction normal to the boundary, close to physical boundary

points, since we must allow two grids whose sides describe the same physical boundary to interpolate from each other in the overlap region; see Figure 3.1.

The interpolation stencil is centered around  $\mathbf{r}_P$  in the parameter plane of the donor grid  $B$ , unless  $\mathbf{r}_P$  is so close to a physical boundary that the centered interpolation formula would use grid points outside of grid  $B$ . In that case, the interpolation stencil is shifted to start at the physical boundary.

If  $\mathbf{r}_P$  is sufficiently far away from all nonphysical boundaries and there are no hole points in the interpolation stencil, we say that the interpolation location is valid.

**4. Inverting a component grid mapping.** Let us consider inverting the mapping corresponding to a general curvilinear grid  $B$  in order to find donor points for an interpolation point with Cartesian coordinate  $\mathbf{x}_P$ . Before we attempt to invert the mapping, we must first determine if  $\mathbf{x}_P \in \Omega^{(B)}$ , since the mapping might not be well defined outside of that domain. If the point is inside of  $\Omega^{(B)}$ , the second problem is to generate a sufficiently good initial approximation for Newton's method, which converges only if the initial guess is sufficiently close to the solution.

**4.1. Boundary mismatch.** The inversion of a component grid mapping can be complicated by mismatch close to a physical boundary. In the present method we handle the mismatch problem by introducing a mismatch tolerance  $\epsilon$  that is used in the following way. During the search for donor points corresponding to an interpolation point that is less than  $\epsilon$  away from a physical boundary in its own component, we say that the interpolation point is inside of the donor grid (for the purpose of interpolation) if it is less than  $\epsilon$  outside of the physical boundary in the donor grid. To avoid errors in the interpolated solutions close to physical boundaries, we compensate for the mismatch when the interpolation data is computed.

For simplicity, we use a global value of  $\epsilon$ . It is desirable to keep  $\epsilon$  as small as possible because the overlap algorithm becomes slower for a larger  $\epsilon$ , since more points get specially treated by the search method. We therefore want to use the smallest  $\epsilon$  that enables all interpolation points close to a physical boundary in their own grid to interpolate from the appropriate donor grid.

The required size of the mismatch tolerance is related to the smoothness of the polygons that represent each part of the global boundary, because each polygon contains points from all grids in an overlap domain. Let there be  $P$  parts of the global boundary, and let the Cartesian coordinates of the sorted grid points of part  $p$  be  $\mathbf{x}_i^{(p)} = (x_i^{(p)}, y_i^{(p)})^T$ ,  $i = 1, 2, \dots, N_p$ . The mismatch tolerance is estimated by

$$(4.1) \quad \epsilon = \max_{1 \leq p \leq P} \left( \max_{2 \leq i \leq N_p - 1} |(\mathbf{x}_{i+1}^{(p)} - \mathbf{x}_i^{(p)}) \cdot \mathbf{n}_i^{(p)}| \right),$$

where the unit normal is given by

$$\mathbf{n}_i^{(p)} = \frac{1}{|\mathbf{x}_i^{(p)} - \mathbf{x}_{i-1}^{(p)}|} \begin{pmatrix} y_i^{(p)} - y_{i-1}^{(p)} \\ -x_i^{(p)} + x_{i-1}^{(p)} \end{pmatrix}.$$

**4.2. Is a point inside a component grid?** To determine if a point with Cartesian coordinate  $\mathbf{x}_P$  is inside a donor grid, we first check if the point is inside of the approximative boundary consisting of the polygon that joins all boundary grid points of the donor grid. This means that the polygonal approximation of the grid boundary is slightly outside the true boundary when it is concave, and slightly inside of the true boundary when it is convex. For points that are inside of the polygon

but outside of the true boundary, the subsequent Newton iteration will converge to a parameter coordinate that is outside of the unit square. If the boundary is physical, it can be determined if the point is within the  $\epsilon$  mismatch tolerance. If the boundary is nonphysical, such a point is classified as being outside of the grid.

The situation is more complicated for points that are outside of the polygon, but inside of the true boundary. When the point is close to nonphysical boundaries in the donor grid or when the point is more than  $\epsilon$  inside of all physical boundaries in its own grid, the problem is not critical because in that case,  $\mathbf{x}_P$  is only slightly inside of the donor grid. According to the requirements for interpolation points, the interpolation location would then be invalid, at least if the gap between the true boundary and the polygon is less than the required overlap.

If the point  $\mathbf{x}_P$  is outside of the polygon but within  $\epsilon$  of a physical boundary in its own component, we must allow interpolation to take place if  $\mathbf{x}_P$  is less than  $\epsilon$  away from a physical boundary in the donor grid. We check this condition by first locating the boundary grid point that is closest to  $\mathbf{x}_P$ . The distance between  $\mathbf{x}_P$  and the closest boundary grid point is decomposed into the normal and tangential components relative to the boundary. If the normal distance is less than  $\epsilon$ , we consider the point  $\mathbf{x}_P$  to be sufficiently close to the boundary for boundary interpolation.

Our technique for checking if  $\mathbf{x}_P$  is inside a polygon is based on counting the intersections between the polygon and a horizontal (constant  $y$ -coordinate) ray that starts at  $\mathbf{x}_P$  and ends at infinity. For this purpose we apply Shimrat's algorithm, which is well known from computational geometry [10].

ALGORITHM 1. *If*

1. *the  $y$ -coordinate of  $\mathbf{x}_P$  is greater than or equal to the minimum value and less than the maximum value of the  $y$ -coordinates of two contiguous vertices of the polygon, then*
2. *the  $x$ -coordinate of the point of intersection is found.*

*If this coordinate is less than the  $x$ -coordinate of point  $\mathbf{x}_P$ , it is counted; otherwise it is not. The test is repeated for all contiguous pairs of vertices. An odd/even number of counts means that  $\mathbf{x}_P$  is inside/outside the polygon.*

*Remark.* The technique can be extended to three space dimensions.

Let the grid points of the donor grid have Cartesian coordinates  $\mathbf{x}_{i,j}$ ,  $1 \leq i \leq N$ ,  $1 \leq j \leq M$ . We define a grid line  $L_{a,b}^{c,d}$ , where either  $a = b$  or  $c = d$  is the polygon that connects the grid points  $\mathbf{x}_{i,j}$  with  $a \leq i \leq b$  and  $c \leq j \leq d$ . It can be verified if a point  $\mathbf{x}_P$  is inside the polygonal approximation of the donor grid boundary, by applying the above algorithm to the polygon consisting of the four grid lines  $L_{1,N}^{1,1}$ ,  $L_{1,N}^{M,M}$ ,  $L_{1,1}^{1,M}$ , and  $L_{N,N}^{1,M}$ .

It is not necessary to traverse through all grid points on the boundary to count the number of intersections with the ray. The number of operations can be reduced by subdividing each side of each component grid in a binary tree structure; see Figure 4.1. Before the overlapping grid algorithm is started, the bounding box of the  $(x, y)$  coordinates of the grid points in the subdivision is saved in each node of the tree.

The ray can only intersect a subdivision of the boundary if it intersects the corresponding bounding box, and the bounding box of each node in the tree contains the union of the bounding boxes of the two subnodes. Hence, it is necessary to search only for intersections in the subnodes whose bounding boxes intersect the ray. We found by experiments that the highest efficiency occurred when each branch of the tree was subdivided recursively until it contained less than five grid points.

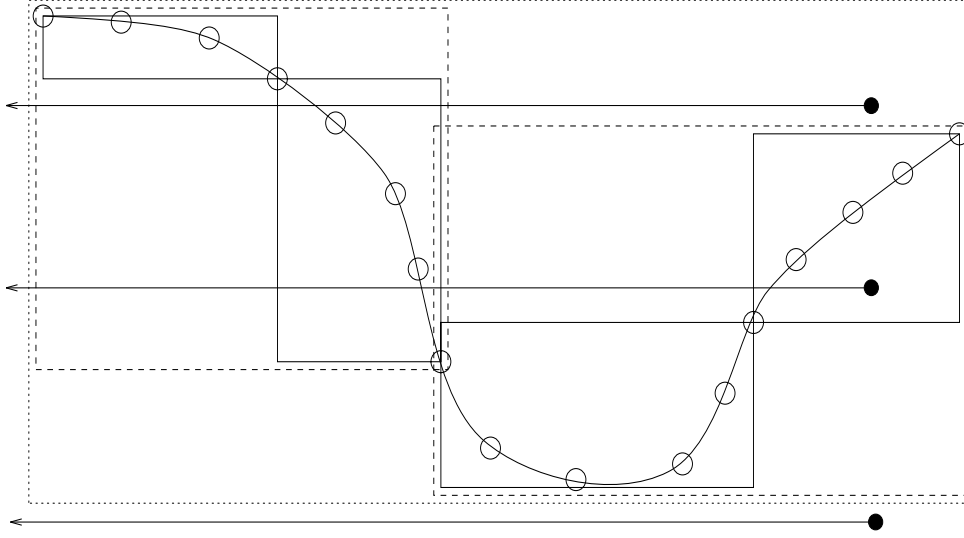


FIG. 4.1. The boundary of each component grid is subdivided in a binary tree structure to speed up the counting of intersections between the ray and the boundary. Open circles indicate grid points on the boundary and filled circles mark different locations of the grid point  $\mathbf{x}_P$ . Also shown are the bounding boxes corresponding to three levels in the binary search tree.

The operational count is as follows. The least expensive case occurs when the ray does not intersect the top level bounding box. This case requires only  $\mathcal{O}(1)$  operations. When the ray intersects the top level bounding box, it is likely to intersect only one of the bounding boxes on each sublevel. There are  $\mathcal{O}(\log_2 N)$  levels on a boundary with  $N$  grid points, so this case requires order  $\mathcal{O}(\log_2 N)$  operations. In the worst case scenario, which is rather unlikely to happen, the boundary oscillates wildly and the ray intersects every subdivision of the boundary. This leads to an operation count of  $\mathcal{O}(N)$ . The total number of operations for determining if a point is inside a component grid follows by summing the effort in counting the number of intersections with the four bounding grid lines. If  $\mathbf{x}_P$  is inside the bounding box of the component grid, the ray must intersect at least one of the top level bounding boxes. By the above argument, the algorithm is most likely to require  $\mathcal{O}(\log_2 N + \log_2 M)$  operations but can, in rare difficult cases, take up to  $\mathcal{O}(N + M)$  operations. We remark that further improvements of the above algorithm are possible; see, for instance, Preparata and Shamos [15].

**4.3. Locating the enclosing grid cell.** If the point  $\mathbf{x}_P$  is found to be inside the grid, we need a good initial guess for Newton's method. A straightforward exhaustive search through all grid points could, for instance, be used to find the closest grid point. However, this approach becomes prohibitively expensive for fine grids, because the number of operations for the search is proportional to the number of grid points. Instead, we apply a bisection technique to locate the enclosing grid cell.

We can assume that  $\mathbf{x}_P$  is inside the grid bounded by the four grid lines  $L_{1,N}^{1,1}$ ,  $L_{1,N}^{M,M}$ ,  $L_{1,1}^{1,M}$ , and  $L_{N,N}^{1,M}$ . To more closely locate  $\mathbf{x}_P$ , we subdivide the grid along the grid lines  $i = [(1 + N)/2]$  and  $j = [(1 + M)/2]$ , which results in four subgrids. By counting the number of intersections between the ray starting at  $\mathbf{x}_P$  and the

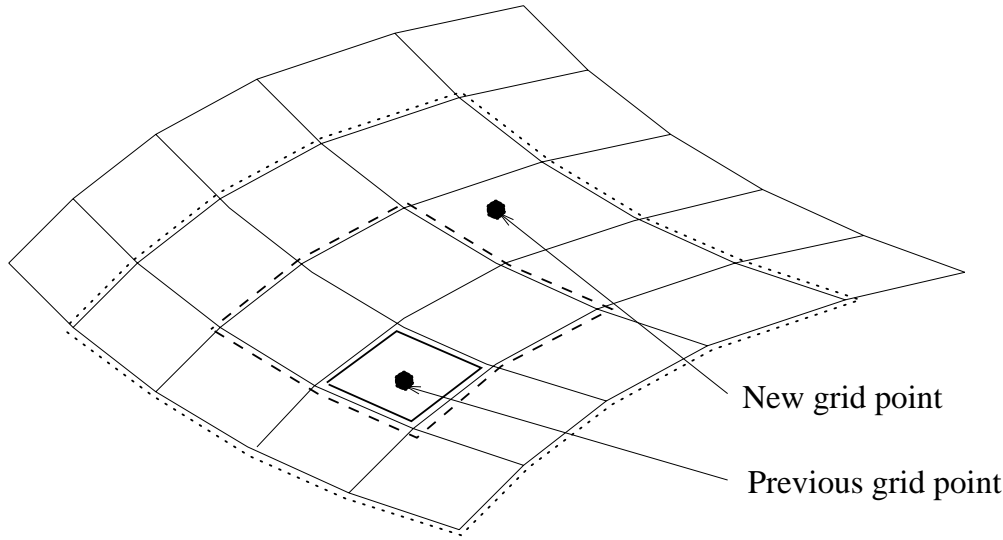


FIG. 4.2. The subgrid is grown by a factor of two around the previous grid point until the new grid point is enclosed by the subgrid. The subgrid is then shrunk until the enclosing grid cell for the new grid point is located.

boundaries of the subgrids, we can determine in which subgrid  $\mathbf{x}_P$  is located. We then repeat the procedure recursively until only one subgrid containing only one grid cell remains. This determines in which grid cell the point  $\mathbf{x}_P$  is located.

Because the grid points are classified sequentially in the overlapping grid algorithm, information about the previous grid point is often available. In this case, the efficiency of the algorithm for locating the enclosing grid cell can be substantially improved. Instead of starting the bisection at the boundary of the donor grid, we then begin by growing a subgrid around the grid cell that enclosed the previous grid point; see Figure 4.2.

We grow the size of the subgrid by a factor of two until the new point is enclosed. The previous bisection technique is then applied to shrink the subgrid down to locate the new enclosing grid cell. Naturally, the new grid point can sometimes be outside of the previous donor grid. To incorporate this case into the algorithm, we limit the growth to an  $8 \times 8$  subgrid. If the new grid point is outside of that subgrid, we treat it as a grid point without an initial guess.

When there is an initial guess for the enclosing grid cell, the new enclosing grid cell can be located in  $\mathcal{O}(1)$  operations. In the absence of an initial guess, approximately  $\log_2(\max(N, M))$  subdivisions of the grid are required to locate the grid cell  $Q_{i,j}$ . Also,  $\mathcal{O}((N + M)/2^q)$  operations are necessary to proceed from subdivision  $q$  to  $q + 1$  if we compute only subdivision information for the boundaries of the component grid. Hence, the operational count becomes  $\mathcal{O}(N + M)$  in the absence of an initial guess.

**4.4. Applying Newton's method.** If the forward mapping is known explicitly, Newton's method can be applied to find the parameter coordinate  $\mathbf{r}_P$  corresponding to  $\mathbf{x}_P$  by taking the initial guess to be the parameter value at the center of the enclosing grid cell. When only the location of the grid points is known, it is necessary to approximate the mapping locally by interpolation before the parameter coordinate

$\mathbf{r}_P$  can be computed. It is consistent to approximate the mapping by a Lagrangian interpolation formula of the same width as when the solution value is interpolated, because both interpolations lead to errors that are of the same order of magnitude. When  $I_W$  is even, the enclosing grid cell uniquely determines the location of the interpolation stencil. Also, when  $I_W$  is odd we increase the width of the interpolation stencil for the mapping by one to make it even. This is harmless, since it only makes the interpolation of the mapping more accurate. Hence, the interpolation formula is always centered around the enclosing grid cell unless a grid boundary forces the interpolation stencil to be skewed. Once the location of the interpolation stencil has been determined, the Lagrangian interpolant is inverted by a Newton iteration to approximate the parameter value  $\mathbf{r}_P$  corresponding to  $\mathbf{x}_P$ .

**5. Classifying the grid points.** Let the overlapping grid have  $G$  component grids, where component  $k$  has  $N_k \times M_k$  grid points. The classification of each grid point  $(i, j)$  in each component grid  $k$  will be stored in the `flag` array according to

$$(5.1) \quad \text{flag}(i, j, k) = \begin{cases} k, & \text{discretization point,} \\ -q, & \text{interpolation point, interpolating from grid } q, \\ 0, & \text{hole point.} \end{cases}$$

To determine which component grid to prefer when there are two or more grids that overlap each other, the component grids are ordered with respect to their priority such that grid  $k$  has priority  $k$ . When there is a choice of which grid points to use in the overlap domain, the basic strategy of the overlapping grid algorithm is to prefer grid points from component grids with higher priority.

We proceed by describing the seven steps that constitute the classification algorithm: hole-cutting, mixed boundary preparation, main classification, boundary mismatch correction, explicit interpolation, consistency check, and trimming. Parallel to the description of the algorithm, we show the result of the different steps for the grid around the NACA-66-006 airfoil in Figure 5.1, which is designed for a fluid flow computation.

In this example the airfoil is described by several components aligned with the body, such that the resolution reflects the curvature of the boundary. The far field is discretized by several Cartesian grids that are fine where the gradient of the velocity field is expected to be large. The airfoil is enclosed in a curved channel to demonstrate how mixed physical/interpolating boundaries are handled.

*Hole-cutting.* The grid points outside of the computational domain are identified in a two-step process related to the technique described by Maple and Belk [8]. We first traverse through all parts of the global boundary to locate all grid cells in all component grids that are intersected by the boundary. We apply the technique described in section 4.2 to determine if each point in the intersected cells is inside or outside of the computational domain by checking each part of the global boundary. Observe that a point is inside only if it is inside relative to all parts of the global boundary. In this way, the points in the intersected cells that are outside of the global boundary are flagged as H-points, while the remaining points in these cells are labeled as G-points. (This notation is not related to (5.1).) We remark that if a part of the global boundary intersects a cell that is less than  $\epsilon$  inside of a physical boundary that belongs to the same part of the global boundary, all grid points in this cell are labeled as G-points. This is done to prevent two overlapping grids, that describe the same physical boundary, from cutting holes in each other in the overlap region. After all parts of the global boundary have been scanned, we identify the hole points by

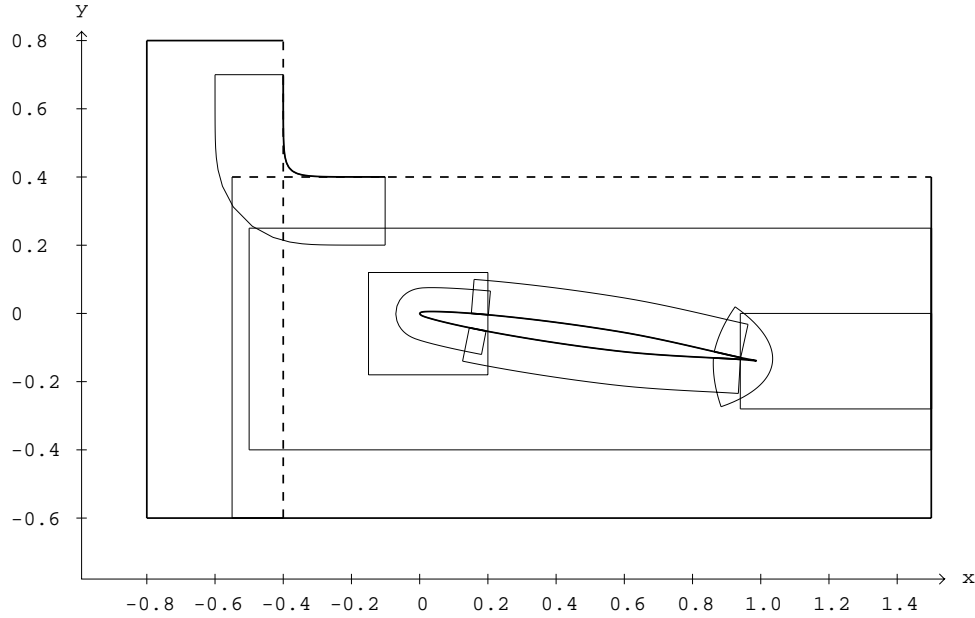


FIG. 5.1. The outline of the components in the airfoil grid. Both parts of the physical boundary are drawn with thick lines, and the thick dashed lines indicate mixed physical/interpolating sides.

starting from the H-points and proceeding along the grid lines in both grid directions. The locations of the G-points next to each H-point define in which direction the hole is situated. The hole extends to the next G-point along the grid line, or until the grid line ends. The result of the hole-cutting is saved in the `flag` array according to

$$\text{flag}(i, j, k) = \begin{cases} 1, & \text{if grid point } (i, j) \text{ in grid } k \text{ is inside,} \\ 0, & \text{otherwise.} \end{cases}$$

The outcome of the hole-cutting algorithm for the airfoil grid is shown in Figure 5.2.

*Mixed boundary preparation.* To correctly identify interpolation points on a mixed interpolating/physical boundary during the main classification algorithm, we go through all grid points on those boundaries and mark the grid points that are sufficiently far inside of another component grid as interpolation boundary points. By sufficiently far inside, we mean at least  $\max(0.5, 0.5(I_W - 2))$  grid cells from the boundary of the donor grid, so that the interpolation location is valid. The grid points on the mixed boundary that are not interpolation boundary points are marked as physical boundary points.

*Main classification.* To more easily present the details of the main classification algorithm, we introduce three functions that implement the rules for discretization and interpolation points described in section 3. Note that we ensure only implicit interpolation at this point.

`interp_from_higher(x, k)` Check if a grid point with Cartesian coordinate  $\mathbf{x}$  can be an interpolation point that interpolates from a donor grid with priority higher than  $k$ . Return the priority of the highest such donor grid or, if  $\mathbf{x}$  could not interpolate from a higher grid, return zero.

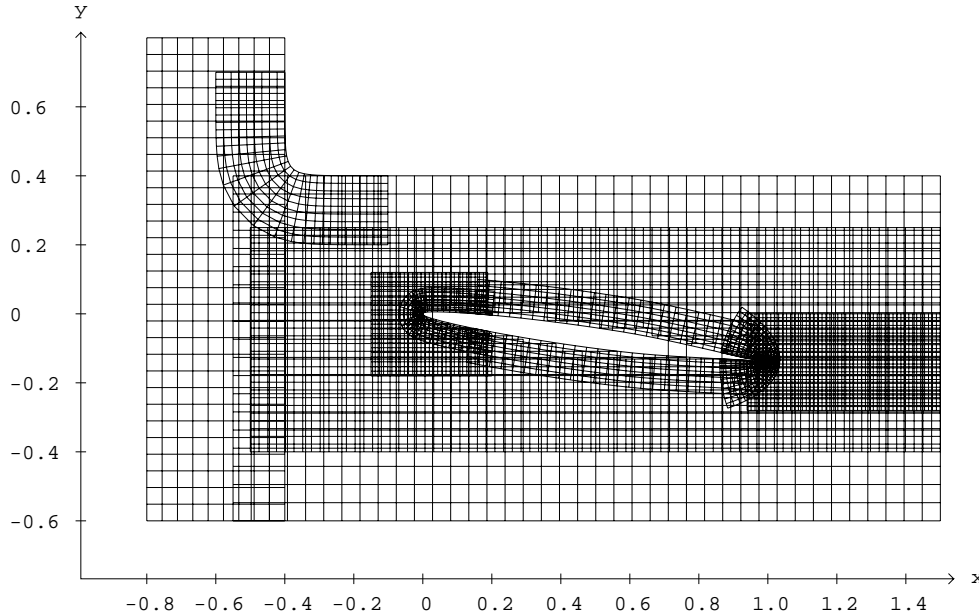


FIG. 5.2. The airfoil grid after all points outside of the computational domain have been removed. The four vertical lines through the airfoil connect contiguous grid points in the coarsest Cartesian background grid which are inside of the computational domain.

`interp_from_lower(x,k)` Proceed as above, but restrict the search to donor grids with priority less than  $k$ .

`discretization_point(i,j,k)` Return TRUE if it is possible to set up the discretization stencil at grid point  $(i,j)$  in grid  $k$ . Otherwise return FALSE.

Pseudo-C-code for the main classification algorithm is presented in Figure 5.3. For each grid point that is not already a hole point, we first investigate if it can interpolate from a grid with higher priority. If this is not possible, we determine if it is a valid discretization point, and if it is not, we instead check if it can interpolate from a lower grid. If it cannot interpolate from a lower grid, the grid point is flagged to be a hole point. The airfoil grid after the main classification is presented in Figure 5.4.

The reason why the main classification can be done in one sweep is that, if the overlap is sufficiently large for implicit interpolation, no points will be classified as hole points during this step. All points that remain after the hole-cutting will therefore be available as donor points. Because only implicit interpolation is ensured at this point, it does not matter if the donor points are discretization points or interpolation points. As long as no point is classified as a hole point, the final result of the main classification is therefore not affected by the order in which the grid points are traversed. The only essential ordering is the priority among the component grids.

After the main classification, the overlapping grid has implicit interpolation and unnecessarily many interpolation points in the overlap regions. These deficiencies will be taken care of in the steps to follow.

*Boundary mismatch correction.* For every interpolation point that is also a physical boundary point, we compute the distance  $\Delta \mathbf{x}$  between the interpolation point and the boundary of the donor grid, normal to the boundary. Let one boundary in-

```

for  $k = G, G - 1, \dots, 1$ 
  for  $i = 1, 2, \dots, N_k$ 
    for  $j = 1, 2, \dots, M_k$ 

/* Do not alter the points that were classified as hole points in the previous steps. */
    if  $\text{flag}(i, j, k) \neq 0$  then
       $\text{interpolee} = \text{interp\_from\_higher}(\mathbf{x}_{i,j}^{(k)}, k);$ 
      if  $\text{interpolee} \neq 0$  then
         $\text{flag}(i, j, k) = \text{interpolee};$ 
      else if  $\text{discretization\_point}(i, j, k)$  then
         $\text{flag}(i, j, k) = k;$ 
      else
         $\text{flag}(i, j, k) = \text{interp\_from\_lower}(\mathbf{x}_{i,j}^{(k)}, k);$ 
      end if
    end if
  end for
end for
end for

```

FIG. 5.3. Main classification algorithm in pseudo-C-code.

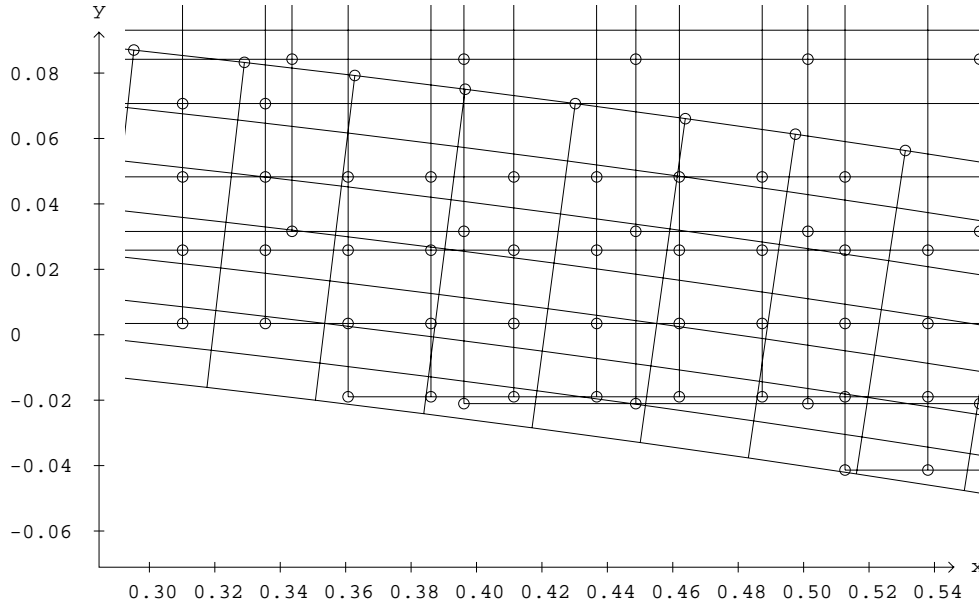


FIG. 5.4. A closeup of the airfoil grid in the vicinity of the upper side of the airfoil, after a successful main classification. Each interpolation point is marked with a circle.

terpolation point have the Cartesian coordinate  $\mathbf{x}_P$ . To compensate for the boundary mismatch we recompute the interpolation location and the donor parameter value by inverting the mapping for the donor grid with the modified coordinate  $\mathbf{x}_P - \Delta \mathbf{x}$ . Since the mismatch can be larger than the grid size normal to the boundary, it is

```

for  $k = G, G - 1, \dots, 1$ 

    for this_interp_point = each interpolation point in grid  $k$ 
        donor_grid = donor grid for this_interp_point;
        do
            new_donor = FALSE; implicit = FALSE;
            for  $(i, j)$  = index of each donor point for this_interp_point
                /* Check if this interpolation is implicit, i.e., if the donor point is an
                interpolation point. */
                if  $\text{flag}(i, j, \text{donor\_grid}) \neq \text{donor\_grid}$ 
                    then
                        /* Check if the donor point can be a discretization point instead. */
                        if (not implicit) and  $\text{discretization\_point}(i, j, \text{donor\_grid})$ 
                            then
                                 $\text{flag}(i, j, \text{donor\_grid}) = \text{donor\_grid}$ ;
                            else
                                implicit = TRUE;
                            end if
                        end if
                    end for
                /* The interpolation point is still implicit. */
                if implicit then
                     $(i, j)$  = index of this_interp_point;
                /* Try to find a donor grid with a lower priority. */
                 $\text{donor\_grid} = \text{interp\_from\_lower}(\mathbf{x}_{i,j}^{(k)}, \text{donor\_grid})$ ;
                /* Try to reclassify the interpolation point  $(i, j)$  into a discretization point. */
                if  $\text{donor\_grid} < k$  and  $\text{discretization\_point}(i, j, k)$  then
                     $\text{flag}(i, j, k) = k$ ;
                else if  $\text{donor\_grid} > 0$  then
                     $\text{flag}(i, j, k) = \text{donor\_grid}$ ;
                    new_donor = TRUE;
                end if
            end if
        /* Check the new donor points for implicit interpolation. */
        while (new_donor);
        end for
    end for

```

FIG. 5.5. *Pseudo-C-code for removing implicit interpolation points.*

necessary to correct the interpolation data for more interpolation points than those on the boundary itself. We therefore recompute the interpolation location and the donor parameter value in a corresponding way for all interpolation points along the grid line that starts at  $\mathbf{x}_P$  and tends to the computational domain. Note that the Cartesian coordinates of the interpolation points are not modified by this procedure, so the smoothness of the grid is not affected. The importance of the mismatch correction is demonstrated by a separate example in section 6.

We remark that the correction method must be improved to handle the case

when a component grid has two physical boundaries on opposite sides, as in Figure 3.1. In this case, the corrections from the two sides could be reconciled according to  $\Delta \mathbf{x}(t) = (1 - t)\Delta \mathbf{x}_0 + t\Delta \mathbf{x}_1$ , where  $t$  is the normalized arclength along the grid line and  $\Delta \mathbf{x}_k$ ,  $k = 0, 1$ , are the corrections at each end of the grid line. Another situation where the correction method must be improved occurs when the interpolation points do not belong to the same grid line. This could, for instance, happen when the boundary-fitted components are nonorthogonal. The correction  $\Delta \mathbf{x}$  would then have to be different for each grid line, but could be computed by following the grid line to the physical boundary and proceeding as before.

*Explicit interpolation.* The next step is to eliminate any implicit interpolations, so this step is disregarded if the interpolation type is implicit. We present the algorithm in pseudo-C-code in Figure 5.5. We start with the highest grid and proceed in decreasing priority order such that fewer restrictions are enforced on the higher grids. For each interpolation point, we determine if the interpolation is implicit or explicit by checking if any of the donor points are interpolation points in the donor grid. If the interpolation is explicit, we proceed to the next interpolation point. Otherwise, we first try to reclassify the donor points that are interpolation points in the donor grid. If all these donor points can be reclassified to be discretization points in the donor grid, the interpolation point becomes explicit, and we proceed to the next interpolation point. However, if it is not possible to reclassify all of them, the interpolation point is flagged as an implicit point. In this case, the algorithm continues by first investigating if the interpolation point can interpolate from a donor grid with higher priority than that of the interpolation point's grid, but lower than the present donor grid. If this fails, we check if it can be a discretization point. If this is not the case, we instead attempt to have the interpolation point interpolate from a donor grid with lower priority than that of the interpolation point's grid. Finally, if not even this can be done, the implicit interpolation is impossible to make explicit. Observe that if the interpolation point was assigned to interpolate from a new donor grid, it is necessary to reiterate and check if the new donor points cause the interpolation to be implicit.

The airfoil grid after the explicit step is presented in Figure 5.6.

*Consistency check.* At this point, it is appropriate to check if the classification of the grid points is consistent, i.e., if all discretization and interpolation points satisfy the necessary requirements. For example, the overlapping grid might not be consistent where two very coarse grids overlap each other. The consistency step will mark all points that fail to satisfy the requirements and will label them according to the rule they fail to fulfill.

To show what happens if there are too few points in the overlap region, we made the inner Cartesian grid around the wing coarser and made the boundary-fitted grid on the upper side of the wing thinner. The graphical representation of the error message is shown in Figure 5.7.

If the overlapping grid is found to be consistent, we proceed to the next step, where all unnecessary interpolation points in the overlap regions are trimmed away.

*Trimming.* The philosophy behind the trimming step is to minimize the number of interpolation and discretization points in the overlapping grid. We will therefore aim at reclassifying interpolation points into hole points. The trimming algorithm employed here is very similar to the trimming step in the method by Chesshire and Henshaw [4].

When the interpolation type is implicit, before the trimming step we must inspect each interpolation point in each grid to check if its donor grid has lower priority than

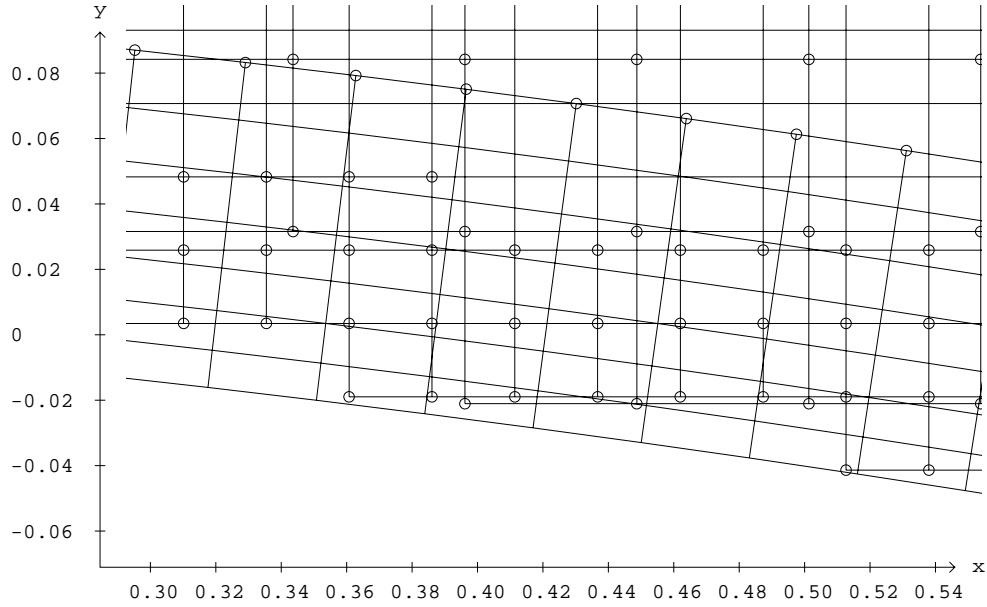


FIG. 5.6. A closeup of the grid in the vicinity of the upper side of the airfoil, after the explicit step has been performed. Each interpolation point is marked with a circle. Note that there are fewer interpolation points in this grid than in Figure 5.4.

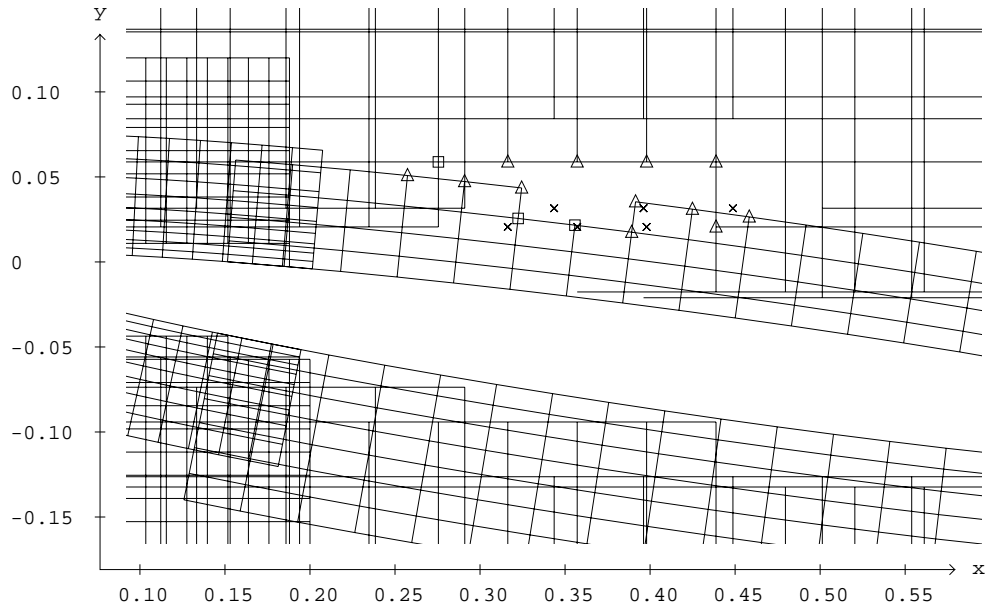


FIG. 5.7. A closeup of the airfoil after an unsuccessful main classification. Bad discretization points are marked with boxes, and hole points that are used as donor points are labeled with crosses. Triangles indicate bad interpolation points because of a hole point among the donor points.

```

for  $k = 1, 2, \dots, G$ 

  for this_interp_point = each interpolation point in grid  $k$ 
     $(i, j)$  = index of this_interp_point;
    if interp_type == explicit or not marked( $i, j, k$ ) then
      if not needed_by_disc( $i, j, k$ ) and not needed_by_interp( $i, j, k$ ) then
        flag( $i, j, k$ ) = 0;
      end if
    end if
  end for

  for  $(i, j)$  = index of each interpolation point in grid  $k$ 
    if discretization_point( $i, j, k$ ) then
      flag( $i, j, k$ ) =  $k$ ;
    end if
  end for

  if interp_type == implicit then
    for this_interp_point = each interpolation point in grid  $k$ 
      if donor_grid(this_interp_point) >  $k$  then
        for  $(i, j)$  = index of each donor point for this_interp_point;
          marked( $i, j, \text{donor\_grid}$ ) = TRUE;
        end for
      end if
    end for
  end if
end for

```

FIG. 5.8. *Pseudo-C-code for the trimming algorithm.*

the priority of the interpolation point's grid. In that case we mark the corresponding donor points to make sure that they are not removed during the trimming algorithm. When the interpolation type is explicit, all points can be regarded as unmarked, since the trimming step reclassifies only interpolation points and we know that all donor points are discretization points when the interpolation type is explicit.

To better describe the trimming algorithm, we introduce two functions that will be used in the following pseudocode.

**needed\_by\_disc**( $i, j, k$ ) Return TRUE if the interpolation point  $(i, j)$  in grid  $k$  is needed by a discretization point in grid  $k$ . Otherwise return FALSE. There are three kinds of discretization points, so it is necessary to check if the point  $(i, j)$  is needed by any interior, boundary, or corner points.

**needed\_by\_interp**( $i, j, k$ ) Return TRUE if the interpolation point  $(i, j)$  in grid  $k$  is a donor point for any interpolation point in another grid. Otherwise return FALSE.

The trimming algorithm (see Figure 5.8) starts at the component grid with the lowest priority and proceeds to the component grid with the highest priority. The list of interpolation points for each grid is traversed three times. During the first examination, each unmarked interpolation point is checked. If the present interpolation

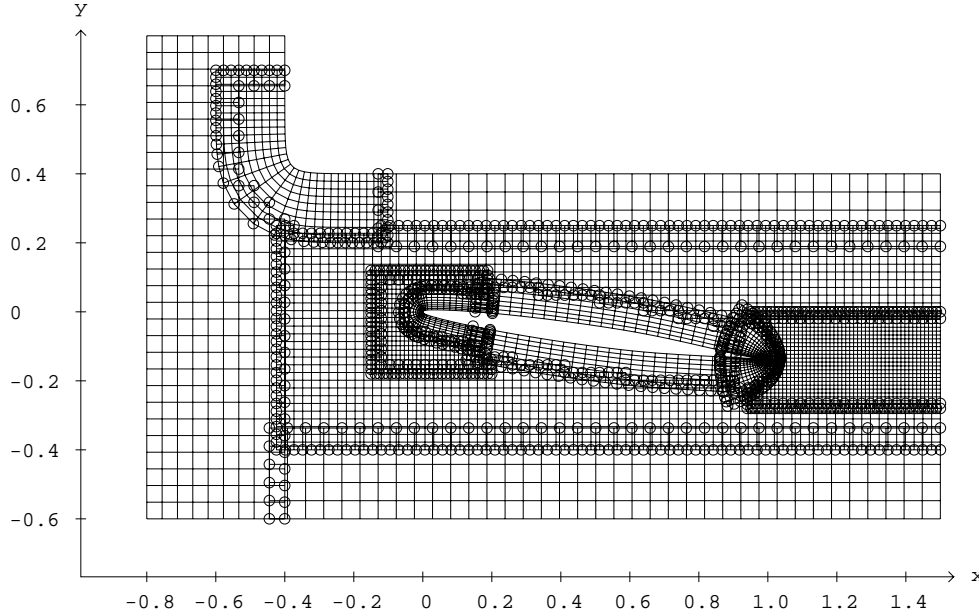


FIG. 5.9. The finished overlapping grid with implicit interpolation. Each interpolation point is marked with a circle.

point is not needed by either a discretization point or another interpolation point, the interpolation point is reclassified to be a hole point. The interpolation points in the present grid are then traversed again. This time we attempt to reclassify all unmarked interpolation points into valid discretization points. During the third and last inspection of the interpolation points, which is done only when the interpolation type is implicit, we mark all donor points if the donor grid has higher priority than the present grid. The purpose of the marking is to make sure that the donor points are not removed when the grids with higher priority are trimmed. The marking is not necessary if the interpolation type is explicit, because in that case, the donor points cannot be interpolation points.

The trimming step completes the overlap algorithm. We present the finished overlapping grid for implicit interpolation in Figure 5.9.

We remark that it is easy to change the trimming algorithm to produce a grid where the size of the overlap is essentially independent of the grid size. To achieve this, we would need only to do the first two substeps of the trimming algorithm in reversed order, i.e., first reclassify as many interpolation points as possible to be discretization points, and thereafter remove the interpolation points that were not needed.

*Performance.* To indicate the CPU-time requirements of the algorithm, we performed some timings. We used the airfoil grid shown above, and a refined grid, where the number of grid points in each grid direction was increased by approximately 50%. The time requirement for ensuring explicit interpolation is comparable to the more involved trimming algorithm for implicit interpolation, so the two types of interpolation require about the same amount of CPU-time. The timings are presented in Table 5.1 and were made with Xcog version 2.0 compiled with optimization and executed on a DEC- $\alpha$  station 200 with 64 Mbyte of RAM.

TABLE 5.1

The CPU-time requirements in seconds to calculate the overlapping grid shown in Figure 5.9 (which has 7265 grid points), and a refined version of the grid. The columns refer to implicit and explicit interpolation types.

| # grid points | Implicit | Explicit |
|---------------|----------|----------|
| 7265          | 1.8      | 1.8      |
| 16312         | 4.0      | 4.1      |

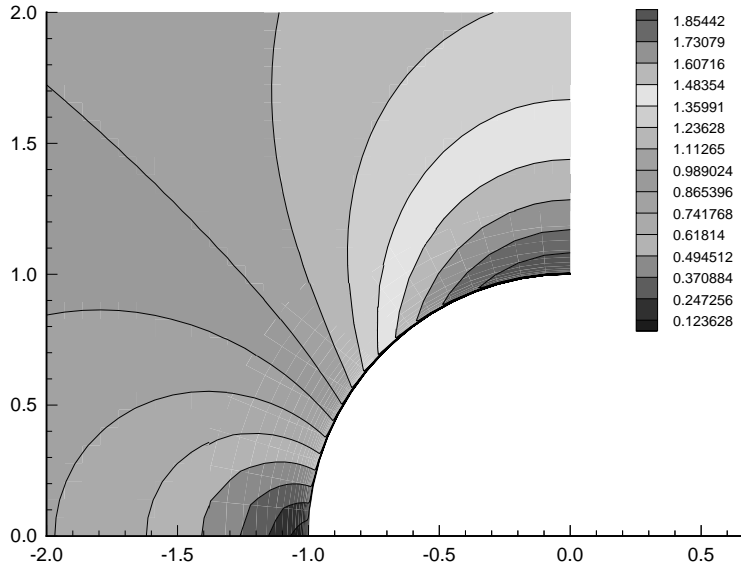


FIG. 6.1. The magnitude of the velocity ( $\sqrt{u^2 + v^2}$ ) for potential flow with a boundary layer at Reynolds number  $10^5$ . The free stream velocity is one, directed to the right, and the boundary layer thickness is approximately  $10^{-2}$ .

This example indicates that the CPU-time usage of the algorithm is of the order of the total number of grid points.

**6. Boundary interpolation.** To investigate the quality of interpolation in a boundary layer, we consider the two-dimensional potential flow around a circular cylinder with radius one, augmented by a boundary layer. When the free stream velocity equals one and is directed in the positive  $x$ -direction, and when the circulation is zero, the velocity field according to potential flow theory satisfies (cf. [18])

$$(6.1) \quad u(x, y) = 1 - \frac{x^2 - y^2}{(x^2 + y^2)^2},$$

$$(6.2) \quad v(x, y) = -\frac{2xy}{(x^2 + y^2)^2}.$$

In the boundary layer close to the boundary of the cylinder, the tangential velocity component is modified according to boundary layer theory. For simplicity, we will use

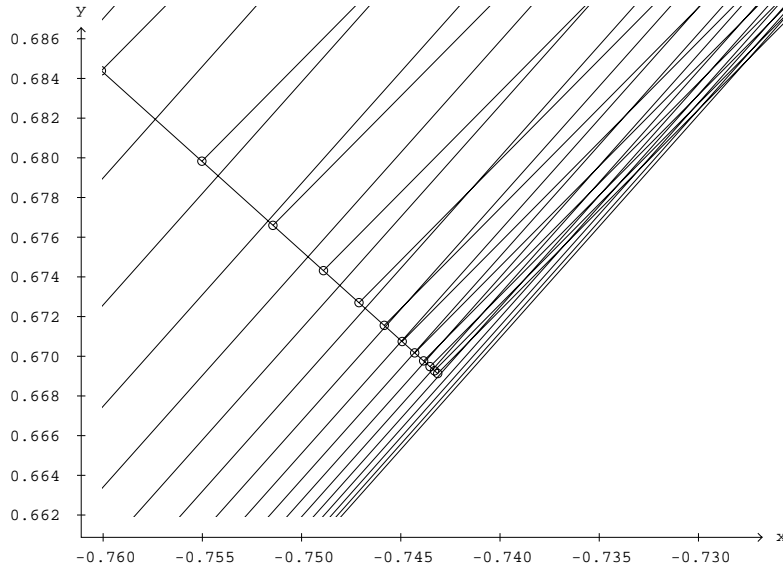


FIG. 6.2. The mismatch at the boundary for the overlapping grid used for the boundary interpolation test. Interpolation points are marked with circles.

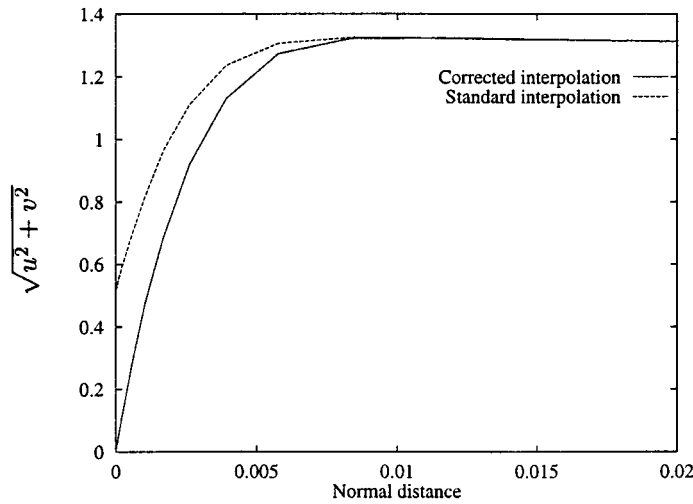


FIG. 6.3. The magnitude of the velocity ( $\sqrt{u^2 + v^2}$ ) for potential flow with a Blasius boundary layer at Reynolds number  $10^5$ , along the interpolation grid line in Figure 6.2. The solid line represents the profile when straightforward interpolation is used and the dashed line corresponds to the profile when the interpolation weights and locations are corrected to account for the mismatch. The corrected and analytical profiles are indistinguishable from each other in this plot.

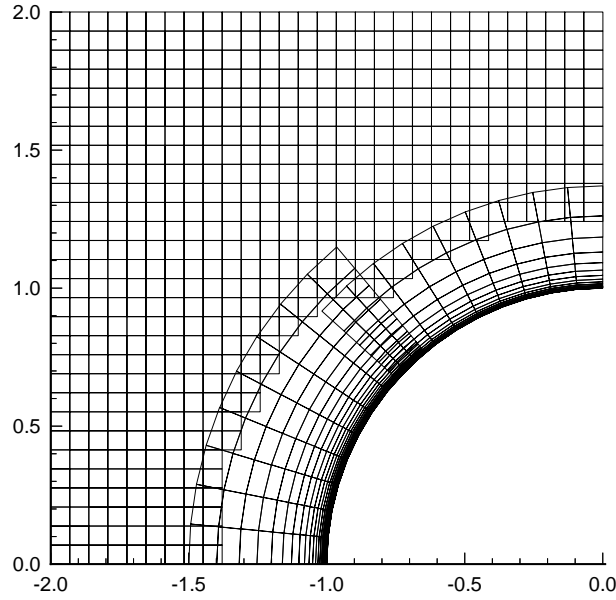


FIG. 6.4. *The overlapping grid used for the boundary interpolation test.*

the profiles provided by Schlichting [16, p. 171], which were calculated by the Blasius series method. The thickness of the boundary layer is proportional to  $1/\sqrt{\text{Re}}$ , where  $\text{Re}$  is the Reynolds number based on the diameter of the cylinder. The magnitude of the velocity field for the case  $\text{Re} = 10^5$  is presented in Figure 6.1. For this Reynolds number, the boundary layer thickness is approximately  $10^{-2}$ .

Let us consider the grid in Figure 6.4. The grid sizes are chosen to reflect the variation of the velocity field. The boundary-fitted grids are stretched in the normal direction and have approximately 10 grid points in the boundary layer. The mappings corresponding to the boundary-fitted grids are defined by bilinear interpolation in between the discrete grid points, and the resulting mismatch in the overlap domain at the boundary is shown in Figure 6.2.

To evaluate the quality of the boundary interpolation, we assign the discretization points in the overlapping grid the velocity field (6.1), (6.2) modified by the boundary layer profile. Thereafter, the velocities at the interpolation points are calculated by biquadratic interpolation from the corresponding donor points. In Figure 6.3, we show the velocity profile along one interpolation grid line, both for standard and corrected interpolation. In this case, the mismatch corresponds to approximately one-tenth of the boundary layer thickness. It can be seen that this mismatch leads to a substantial error in the interpolated velocity when the standard interpolation procedure is applied. However, the error is almost completely removed when the interpolation locations and donor parameter values are corrected for the mismatch. We remark that the mismatch, and therefore the interpolation error, can also be removed by modifying the mappings to follow the boundary exactly.

**7. Conclusions.** A general purpose algorithm for assembling two-dimensional overlapping grid systems has been described. The method produces a grid with as little overlap as possible, both for implicit and explicit interpolation. It also compensates the interpolation data for the mismatch that can occur when a boundary of the computational domain is represented by several overlapping component grids. If the overlap between the components is insufficient, the algorithm reports the inconsistent grid points to the user, to facilitate an improvement of the input. The algorithm has been implemented in the code Xcog, which is distributed free of charge.

The extension to three dimensions is under way and a preliminary version of the algorithm was presented by Malmheden and Petersson [7]. To perform the hole-cutting, the global boundary could be represented by a surface triangulation for each part of the surface, and it would be possible to determine if a point is inside or outside of the computational domain by using the three-dimensional version of the ray method. The specification of the external part of a mixed physical/external side of a component grid would also have to be generalized. This would, for instance, be necessary to handle intersecting surfaces, such as a wing-fuselage configuration. These issues are discussed by Petersson [13].

**Acknowledgments.** I thank the people who patiently tested Xcog during its development. They made valuable suggestions for improvements and located bugs and weaknesses in previous versions of the code. These people include: Per Hammarlund; Johan Malmheden; Lotta, Fredrik, and Pelle Olsson; Olof Runborg; Björn Sjögreen; Jon Tegnér; and Jacob Yström. I would also like to thank David Brown, Geoff Chesshire, Bill Henshaw, and Heinz-Otto Kreiss for initiating my interest in overlapping grids and sharing their thorough knowledge on the subject. Finally, I thank three anonymous referees for pointing out shortcomings in the original manuscript.

#### REFERENCES

- [1] J. A. BENEK, P. G. BUNING, AND J. L. STEGER, *A 3-D Chimera Grid Embedding Technique*, AIAA paper 85-1523, American Institute of Aeronautics and Astronautics, Reston, VA, 1985, pp. 322-331.
- [2] J. A. BENEK, J. L. STEGER, AND F. C. DOUGHERTY, *A Flexible Grid Embedding Technique with Application to the Euler Equations*, AIAA paper 83-1944, American Institute of Aeronautics and Astronautics, Reston, VA, 1983, pp. 373-382.
- [3] D. L. BROWN, G. CHESHIRE, AND W. D. HENSHAW, *Getting started with CMPGRD. Introductory User's Guide and Reference Manual*, Tech. report LA-UR 90-3729, Los Alamos National Laboratory, Los Alamos, NM, 1989.
- [4] G. CHESHIRE AND W. D. HENSHAW, *Composite overlapping meshes for the solution of partial differential equations*, J. Comput. Phys., 90 (1990), pp. 1-64.
- [5] W. D. HENSHAW, *Ogen: An Overlapping Grid Generator for Overture*, Tech. report LA-UR 96-3466, Los Alamos National Laboratory, Los Alamos, NM, 1996.
- [6] B. KREISS, *Construction of a curvilinear grid*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 270-279.
- [7] J. F. MALMHEDEN AND N. A. PETERSSON, *A demonstration of the 3-D overlapping grid code CHALMESH*, in Proceedings of the 3rd Symposium on Overset Composite Grid and Solution Technology, Los Alamos National Laboratory, 1996.
- [8] R. C. MAPLE AND D. M. BELK, *A new approach to domain decomposition: The Beggar code*, in Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, N. P. Weatherill, ed., Pine Ridge Press, 1994, pp. 305-314.
- [9] R. L. MEAKIN, *A New Method for Establishing Intergrid Communication Among Systems of Overset Grids*, AIAA paper, 91-1586-CP, American Institute of Aeronautics and Astronautics, Reston, VA, 1991.
- [10] M. S. MILGRAM, *Does a point lie inside a polygon?*, J. Comput. Phys., 84 (1989), pp. 134-144.

- [11] R. W. NOACK AND D. M. BELK, *Improved interpolation for viscous overset grids*, in Proceedings of the 3rd Symposium on Overset Composite Grid and Solution Technology, Los Alamos National Laboratory, Los Alamos, NM, 1996.
- [12] S. J. PARKS, P. G. BUNING, J. L. STEGER, AND W. M. CHAN, *Collar Grids for Intersecting Geometric Components within the Chimera Overlapped Grid Scheme*, AIAA paper 91-1587-CP, American Institute of Aeronautics and Astronautics, Reston, VA, 1991.
- [13] N. A. PETERSSON, *Hole-cutting for 3-D overlapping grids*, SIAM J. Sci. Comput., to appear.
- [14] N. A. PETERSSON, *User's Guide to Xcog Version 2.0*, Tech. report CHA/NAV/R-97/0048, Hydromechanics Division, Naval Arch. and Ocean Eng., Chalmers Univ. of Tech., Gothenburg, Sweden, 1997. Also at <http://www.na.chalmers.se/~andersp/xcog/xcog.html>.
- [15] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry*, Springer-Verlag, New York, 1985.
- [16] H. SCHLICHTING, *Boundary Layer Theory*, 7th ed., McGraw-Hill, New York, 1979.
- [17] N. E. SUHS, *Tutorial: PEGSUS version 4.0*, in Proceedings of the 2nd Symposium on Overset Composite Grid and Solution Technology, American Institute of Aeronautics and Astronautics, Reston, VA, 1994.
- [18] C.-S. YIH, *Fluid Mechanics*, West River Press, Ann Arbor, MI, 1979.